

Harjoitustyö 2: Game of Taxes

Viimeksi päivitetty 16.03.2018

Huom. koska tämä harjoitustyö pohjautuu aiempaan harjoitustyöhön, on allaolevassa merkitty harmaalla pohjalla kaikki muuttuneet / uudet asiat.

Harjoitustyön aihe

Tässä harjoitustyössä ensimmäisen harjoitustyön ohjelmaa laajennetaan niin, että se osaa käsitellä myös kaupunkien verotussuhteita. Historian edetessä kuvitteellisen maailmamme kaupungit ovat alkaneet tehdä yhteistyötä, ja vahvimmat kaupungit tarjoavat tukeaan heikommille, mutta vaativat vastapainoksi osan ko. kaupungin verokertymästä.

Kaupunkien yksilöiminen tehdään nyt yksikäsitteisen ID-tunnisteen avulla (osoittimien sijaan). Harjoitustyössä paino on valmiiden tietorakenteiden ja algoritmien käytössä (STL), mutta edelleen harjoitellaan myös algoritmien toteuttamista ja niiden tehokkuuden arvioimista.

Harjoitustyössä ohjelmaan syötetään tietoja kaupungeista (nimi ja koordinaatit ja verokertymä), ja ohjelmalta voi kysyä kaupunkia halutussa järjestyksessä sekä minimi- ja maksimitietoja. Ei-pakollisena osana kaupunkia voi myös poistaa. Kysyä voi myös kaupunkien verotussuhteita ja verokertymää.

Koska kyseessä on Tietorakenteiden ja algoritmien harjoitustyö, ohjelman tehokkuus on tärkeä arvostelukriteeri. Tavoitteena on tehdä mahdollisimman tehokas toteutus, kun oletetaan että kaikki ohjelman tuntemat komennot ovat suunnilleen yhtä yleisiä (ellei komentotaulukossa toisin mainita). Plussaa tietysti saa, mitä tehokkaammin operaatiot pystyt toteuttamaan.

Huomaa erityisesti seuraavat asiat:

- Tässä harjoitustyössä harjoitellaan erityisesti valmiiden tietorakenteiden ja algoritmien (STL) tehokasta käyttöä, joten kannattaa suosia STL:ää omien algoritmien/tietorakenteiden sijaan silloin, kun se on tehokkuuden kannalta järkevää.
- Valmiina annetun pääohjelman voi ajaa joko graafisen käyttöliittymän kanssa QtCreatorilla/qmakella käännettynä, tai tekstipohjaisena pelkällä g++:lla käännettynä. Itse ohjelman toiminnallisuus ja opiskelijan toteuttama osa on täsmälleen sama molemmissa tapauksissa.
- **Vihje** tehokkuudesta: Jos minkään operaation keskimääräinen tehokkuus on huonompi kuin $\Theta(n \log n)$, ratkaisun tehokkuus ei ole hyvä. Suurin osa operaatioista on mahdollista toteuttaa paljon nopeamminkin.

- Osana ohjelman palautusta tiedostoon `datastructures.hh` on jokaisen operaation oheen laitettu kommentti, johon lisätään oma arvio kunkin toteutetun operaation asympotoottisesta tehokkuudesta lyhyiden perusteluiden kera.
- Osana ohjelman palautusta palautetaan git:ssä myös dokumentti nimeltä `"readme.pdf"` (hakemistossa `"prg2"`), jossa perustellaan toteutuksessa käytetyt tietorakenteet tehokkuuden kannalta.
- Operaatioiden `remove()`, `towns_distance_increasing_from()`, `longest_vassal_path()` ja `total_net_tax()` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua, joten toteuttamatta jättäminen vaikuttaa harjoitustyön arvosanaan! **Vain pakolliset osat toteuttamalla harjoitustyön maksimi-arvosana on 3.**
- Tehokkuudessa olennaisinta on, miten ohjelman tehokkuus muuttuu datan kasvaessa, eivät pelkät sekuntimäärät.
- Operaation tehokkuuteen lasketaan kaikki siihen liittyvä työ, myös mahdollisesti alkioden lisäyksen yhteydessä tehty operaation hyväksi liittyvä työ.
- Plussaa tietysti saa, jos operaatioita saa toteutettua vaadittua mahdollisimman tehokkaasti.
- Samoin plussaa saa, mitä nopeammaksi operaatiot saa sekunteinakin (jos siis kertaluokka on vähintään vaadittu). **Mutta** plussaa saa vain tehokkuudesta, joka syntyy omista algoritmivalinnoista ja suunnittelusta. (Esim. kääntäjän optimointivipujen vääntely, rinnakkaisuuden käyttö, häkkerioptimoinnilla kellojaksojen viilaaminen eivät tuo pisteitä.)
- Riittävän huonolla toteutuksella työ voidaan hylätä.
- Esimerkkejä kysymyksistä, joilla tehokkuutta voi usein parantaa: Tehdäänkö jokin asia turhaan useaan kertaan? Voiko jonkin asian joskus jättää kokonaan tekemättä? Tehdäänkö joissain työtä enemmän kuin on välttämättä tarpeen? Voiko jonkin asian tehdä "lähes ilmaiseksi" samalla, kun tehdään jotain muuta?

Kaupunkien etäisyyksistä

Joissain operaatioissa vertaillaan kaupunkien etäisyyksiä (joko origosta (0,0) tai annetusta koordinaatista). Jotta automaattitesteissä ei jouduttaisi ongelmiin liukulukujen kanssa, sovitaan että tämän harjoitustyön maailmassa on mahdollista liikkua vain yksi yksikkö kerrallaan itään, länteen, pohjoiseen tai etelään (ei siis vinoon). Tällöin kahden pisteen välisen etäisyyden voi laskea yksinkertaisesti kaavalla $|x_1 - x_2| + |y_1 - y_2|$, eli koordinaattien erotusten itseisarvon summana (kiinnostuneille, tällaista geometriaa kutsutaan nimellä *Manhattan/taxicab geometry*).

Verokertymän laskemisesta

Harjoitustyössä kaupungit maksavat toisille kaupungeille veroa. Kaupunkia, joka maksaa veroa toiselle kaupungille, kutsutaan veroja saavan kaupungin *vasallikaupungiksi* (*vassal town*). Vastaavasti veroja vasallikaupungilta saavaa kaupunkia kutsutaan *isäntäkaupungiksi* (*master town*).

Jokaiselle kaupungille määritellään kaupunkia lisättäessä sen vuotuinen kansalaisilta saatava vero (kokonaislukuna). Tämän lisäksi kaupunki saa 10 % jokaisen vasallikaupunkinsa kokonaisverokertymästä (mukaanlukien vasallikaupungin mahdollisesti omilta vasalleiltaan saamat verot). Tästä kertymästä (kansalaisilta saatava määrä + vasalleilta saatava kymmenys) kaupunki maksaa sitten vielä veroa omalle isäntäkaupungilleen (jos sellainen on). Laskuissa 10 % laskeminen pyöristetään aina alaspäin (kuten C++:n kokonaislukujen jakolasku tekee), ja tämä alas pyöristetty summa vähennetään sitten verokertymästä. Tuloksena saatavaa lukua kutsutaan tässä *nettoverokertymäksi (total net tax)*.

Esimerkki: kaupungilla x (vero 20) on kaksi vasallia $v1$ (vero 10) ja $v2$ (vero 5), joilla ei ole omia vasalleja. Lisäksi kaupungilla x on isäntäkaupunki i . Tällöin kaupungin $v1$ nettoverokertymä on 9 (koska 10 % maksetaan verona x :lle), kaupungin $v2$ nettoverokertymä on 5 (koska 10 % 5:stä pyöristyy nollassa). Kaupungin x nettoverokertymä on 19 ($20 + v1$:ltä saatava 1 (ja $v2$:lta saatava 0) = 21, josta 10 % eli pyöristettynä 2 maksetaan i :lle, tekee yhteensä 19).

Järjestämisestä

Kaupungeja järjestettäessä on mahdollista, että etäisyyden mukaan järjestettäessä usealla on sama etäisyys (tai nimijärjestyksessä nimi). Tällaisten tapausten keskinäinen järjestys on mielivaltainen.

Ohjelman hyväksymissä nimissä voi olla vain kirjaimia A-Z ja a-z. Järjestämisen voi tehdä joko `std::string`-luokan vertailuoperaattorin "`<`" mukaan (jossa isot kirjaimet tulevat ennen pieniä) tai "oikealla tavalla", jossa vastaavat isot ja pienet kirjaimet ovat samanarvoisia.

Kaupunkien ID:iden järjestämisessä tulee käyttää C++:n `string`-luokan "`<`"-vertailua.

Harjoitustyön toteuttamisesta ja C++:n käytöstä

Harjoitustyön kielenä on C++14. Tämän harjoitustyön tarkoituksena on opetella valmiiden tietorakenteiden ja algoritmien käyttöä, joten C++:n STL:n käyttö on erittäin suotavaa ja osa arvostelua. Mitään erityisiä rajoituksia C++:n standardikirjaston käytössä ei ole. Luonnollisesti kielen ulkopuolisten kirjastojen käyttö ei ole sallittua (esim. Windowsin omat kirjastot tms.).

HUOMAA, että koska tämän harjoitustyön tarkoituksena on harjoitella STL:n käyttöä, on *erittäin todennäköistä*, että ensimmäisen harjoitustyön tietorakenneratkaisusi EI ole paras mahdollinen tässä harjoitustyössä. Samoin kannattaa harkita, missä kohdin ensimmäisessä harjoitustyössä itse toteutettuja algoritmeja voi korvata STL:n valmiilla algoritmeilla!

Ohjelman toiminta ja rakenne

Osa ohjelmasta tulee valmiina kurssin puolesta, osa toteutetaan itse.

Valmiit osat, jotka tarjotaan kurssin puolesta

Tiedostot `mainprogram.hh`, `mainprogram.cc`, `mainwindow.hh`, `mainwindow.cc`, `mainwindow.ui` (joihin **EI SAA TEHDÄ MITÄÄN MUUTOKSIA**)

- Pääohjelma, joka hoitaa syötteiden lukemisen, kommentojen tulkitsemisen ja tulostusten tulostamisen. Pääohjelmassa on myös valmiina komentoja testaamista varten.

- QtCreatorilla tai qmakella käännettäessä graafinen käyttöliittymä, jonka "komentotulkkiin" voi näppäimistön lisäksi hiirellä lisätä komentoja, tiedostoja yms. Graafinen käyttöliittymä näyttää myös luodut kaupungit ja niiden vasallisuhteet graafisesti samoin kuin suoritettujen operaatioiden tulokset.

Tiedosto *datastructures.hh*

- `class Datastructures`: Luokka, johon harjoitustyö kirjoitetaan. Luokasta annetaan valmiina sen julkinen rajapinta (johon **EI SAA TEHDÄ MITÄÄN MUUTOKSIA**)
- Tyypimmääritys `TownID`, jota käytetään kaupungit yksilöivänä tunnisteena (samannimisiä kaupunkeja voi olla monta ja vaikka samoissa koordinaateissa, mutta jokaisella on eri id).
- Vakiot `NO_ID`, `NO_NAME` ja `NO_VALUE`, joita käytetään paluuarvoina, jos tietoja kysytään kaupungista, jota ei ole olemassa.

Tiedosto *datastructures.cc*

- Tähän luonnollisesti kirjoitetaan luokan operaatioiden toteutukset.
- Funktio `random_in_range`: Arpoo luvun annetulla välillä (alku- ja loppuarvo ovat molemmat välissä mukana). Voit käyttää tätä funktiota, jos tarvitset toteutuksessasi satunnaislukuja.

Graafisen käyttöliittymän käytöstä

Tässä harjoitustyössä graafisesta käyttöliittymästä (QtCreatorilla käännettäessä) on enemmän hyötyä testaamisessa, koska kaupunkien sijaintien lisäksi käyttöliittymä näyttää kaupunkien väliset verotussuhteet. Lisäksi käyttöliittymää on laajennettu niin, että kaupungin klikkaaminen hiirellä tulostaa sen tiedot (kuten ennenkin), mutta myös kopioi kaupungin ID:n komentoriville (kätevä tapa syöttää komentojen parametreja).

Harjoitustyönä toteutettavat osat

Tiedostot *datastructures.hh* ja *datastructures.cc*

- `class Datastructures`: Luokan julkisen rajapinnan jäsenfunktiot tulee toteuttaa. Luokkaan saa listätä omia määrittelyitä (jäsenmuuttujat, uudet jäsenfunktiot yms.)
- Tiedostoon *datastructures.hh* kirjoitetaan jokaisen toteutetun operaation yläpuolelle kommentteihin oma arvio ko. operaation toteutuksen asympotoottisesti tehokkuudesta ja lyhyt perustelu arviolle.

Lisäksi harjoitustyönä toteutetaan alussa mainittu dokumentti *readme.pdf*.

Huom! Omassa koodissa ei ole tarpeen tehdä ohjelman varsinaiseen toimintaan liittyviä tulostuksia, koska pääohjelma hoitaa ne. Mahdolliset Debug-tulostukset kannattaa tehdä cerr-virtaan (tai `qDebug`:lla, jos käytät Qt:ta), jotta ne eivät sotke testejä.

Ohjelman tuntemat komennot ja luokan julkinen rajapinta

Kun ohjelma käynnistetään, se jää odottamaan komentoja, jotka on selitetty alla. Komennot, joiden yhteydessä mainitaan jäsenfunktio, kutsuvat ko. Datastructure-luokan operaatioita, jotka siis opiskelijat toteuttavat. Osa komennoista on taas toteutettu kokonaan kurssin puolesta pääohjelmassa.

Jos ohjelmalle antaa komentoriviltä tiedoston parametriksi, se lukee komennot ko. tiedostosta ja lopettaa sen jälkeen.

Komento Julkinen jäsenfunktio	Selitys
add_town id nimi (x,y) vero bool add_town(TownID id, std::string const& name, int x, int y, int tax);	Lisää tietorakenteeseen uuden kaupungin annetulla uniikilla id:llä, nimellä, sijainnilla ja verotulolla. Aluksi kaupungit eivät ole minkään toisen kaupungin vasalleja. Jos annetulla id:llä on jo kaupunki, ei tehdä mitään ja palautetaan false , muuten palautetaan true .
remove id bool remove_town(TownID id);	Poistaa annetulla id:llä olevan kaupungin. Jos id ei vastaa mitään kaupunkia, ei tehdä mitään ja palautetaan false , muuten palautetaan true . Jos poistettavalla kaupungilla on vasallikaupunkeja ja isäntäkaupunki, poistettavan kaupungin vasallit siirtyvät poistettavan kaupungin isännän vasalleiksi. Jos poistettavalla ei ole isäntää, poistettavan mahdolliset vasallitkin jäävät ilman isäntää poiston jälkeen. <i>Tämän operaation tehokkuus ei ole kriittisen tärkeää (sitä ei oleteta kutsuttavan usein), joten se ei ole oletuksena mukana tehokkuustesteissä. Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).</i>
add_vassalship vassalid masterid bool add_vassalship(TownID vassalid, TownID masterid);	Lisää kaupunkien välille vasallisuhteen. Kaupunki voi olla vain yhden kaupungin vasallikaupunki. Työssä saa olettaa, että vasallisuhteet eivät voi muodostaa silmukoita (ts. kaupunki ei voi olla suoraan tai epäsuorasti itsensä vasalli). Jos jompaa kumpaa kaupunkia ei löydy tai vasallilla on jo isäntä, ei tehdä mitään ja palautetaan false . Muuten palautetaan true .
(no command) std::string get_name(TownID id);	Palauttaa annetulla ID:llä olevan kaupungin nimen, tai NO_NAME, jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) <i>Tätä operaatiota kutsutaan useammin kuin muita.</i>
(no command) std::pair<int, int> get_coordinates(TownID id);	Palauttaa annetulla ID:llä olevan kaupungin sijainnin, tai parin (NO_VALUE, NO_VALUE), jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) <i>Tätä operaatiota kutsutaan useammin kuin muita.</i>

Komento Julkinen jäsenfunktio	Selitys
(no command) <code>int get_tax(TownID id);</code>	Palauttaa annetulla ID:llä olevan kaupungin verotiedon, tai NO_VALUE, jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) <i>Tätä operaatiota kutsutaan useammin kuin muita.</i>
vassals id <code>std::vector<TownID> get_vassals(TownID id);</code>	Palauttaa annetulla ID:llä olevan kaupungin välittömien vasallien id:t (ts. mukaan <i>ei lasketa</i> vasallien vasalleja), tai vektorin jonka ainoa alkio on NO_ID, jos id:llä ei löydy kaupunkia. Paluuarvo on järjestettävä nousevan ID:n mukaiseen järjestykseen. (Pääohjelma kutsuu tätä eri paikoissa.)
size <code>unsigned int size();</code>	Palauttaa tietorakenteessa olevien kaupunkien lukumäärän.
clear <code>void clear();</code>	Tyhjentää tietorakenteet eli poistaa kaikki kaupungit (tämän jälkeen size palauttaa 0).
find nimi <code>std::vector<TownID> find_towns(std::string const& name);</code>	Palauttaa kaupungit, joilla on annettu nimi tai tyhjän vektorin, jos sellaisia ei ole. Paluuarvo on järjestettävä nousevan ID:n mukaiseen järjestykseen. <i>Tätä operaatiota kutsutaan harvoin, eikä se ole oletuksena mukana tehokkuustestissä.</i>
change_name id newname <code>bool change_town_name(TownID id, std::string const& newname);</code>	Muuttaa annetulla ID:llä olevan kaupungin nimen. Jos kaupunkia ei löydy, palauttaa <code>false</code> , muuten <code>true</code> .
all_towns <code>std::vector<TownID> all_towns();</code>	Palauttaa kaikki tietorakenteessa olevat kaupungit mielivaltaisessa järjestyksessä. <i>Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.</i>
alphalist <code>std::vector<TownID> towns_alphabetically();</code>	Palauttaa kaupungit nimen mukaan aakkosjärjestyksessä.
distlist <code>std::vector<TownID> towns_distance_increasing();</code>	Palauttaa kaupungit etäisyysjärjestyksessä origosta (0,0), lähin ensin. <i>Huomaa etäisyyden määritelmä aiempana.</i>
mindist <code>TownID min_distance();</code>	Palauttaa kaupungin, joka on lähinnä origoa (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos kaupunkeja ei ole, palautetaan NO_ID. <i>Huomaa etäisyyden määritelmä aiempana.</i>
maxdist <code>TownID max_distance();</code>	Palauttaa kaupungin, joka on kauimpana origosta (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos kaupunkeja ei ole, palautetaan NO_ID. <i>Huomaa etäisyyden määritelmä aiempana.</i>
nth_distance n <code>TownID nth_distance(unsigned int n);</code>	Palauttaa kaupungin, joka on etäisyysjärjestyksessä n:s origosta (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos <i>n</i> on 0 tai suurempi kuin kaupunkien määrä, palautetaan NO_ID. <i>Huomaa etäisyyden määritelmä aiempana.</i>

Komento Julkinen jäsenfunktio	Selitys
towns_from (x,y) std::vector<TownID> towns_distance_increasing_from (int x, int y);	Palauttaa kaupungit etäisyysjärjestyksessä annetusta koordinaatista (x,y), lähin ensin. <i>Huomaa etäisyyden määritelmä aiempaan. Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).</i>
taxer_path id std::vector<TownID> taxer_path (TownID id);	Palauttaa listan kaupungeista, joille kaupunki maksaa veroa suoraan tai epäsuorasti. Paluuvektoriin talletetaan ensin kaupunki itse, sitten sen isäntä, isännän isäntä jne. niin kauan kuin isäntiä riittää. Jos id:llä ei ole kaupunkia, palautetaan tyhjä vektori.
longest_vassal_path id std::vector<TownID> longest_vassal_path (TownID id);	Palauttaa pisimmän mahdollisen ketjun vasalleja kaupungista lähtien. Paluuvektoriin talletetaan ensin kaupunki, sitten kaupungin vasalli, vasallin vasalli jne. niin, että ketjussa on mahdollisimman monta kaupunkia (jos yhtä pitkiä ketjuja on useita, palautetaan jokin niistä). Jos id:llä ei ole kaupunkia, palautetaan tyhjä vektori. <i>Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).</i>
total_net_tax id int total_net_tax (TownID id);	Palauttaa kaupungin nettoverokertymän (joka on määriteltä tässä ohjeessa aiemmin). Jos id:llä ei löydy kaupunkia, palautetaan NO_VALUE. <i>Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).</i>
random_add n (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) n kpl kaupunkeja, joilla on satunnainen id, nimi, sijainti ja vero. 80 % todennäköisyydellä kaupunki lisätään myös toisen vasalliksi. Huom! Arvot ovat tosiaan satunnaisia, eli saattavat olla kerrasta toiseen eri.
random_seed n (pääohjelman toteuttama)	Asettaa pääohjelman satunnaislukugeneraattorille uuden siemenarvon. Oletuksena generaattori alustetaan joka kerta eri arvoon, eli satunnainen data on eri ajokerroilla erilaista. Siemenarvon asettamalla arvotun datan saa toistumaan samanlaisena kerrasta toiseen (voi olla hyödyllistä debuggaamisessa).
read 'tiedostonimi' (pääohjelman toteuttama)	Lukee lisää komentoja annetusta tiedostosta. (Tällä voi esim. lukea listan tiedostossa olevia työntekijöitä tietorakenteeseen, ajaa valmiita testejä yms.)
stopwatch on / off / next (pääohjelman toteuttama)	Aloittaa tai lopettaa komentojen ajanmittauksen. Ohjelman alussa mittaus on pois päältä ("off"). Kun mittaus on päällä ("on"), tulostetaan jokaisen komennon jälkeen siihen kulunut aika. Vaihtoehto "next" kytkee mittauksen päälle vain seuraavan komennon ajaksi (kätevää read-komennon kanssa, kun halutaan mitata vain komentotiedoston kokonaisaika).

Komento Julkinen jäsenfunktio	Selitys
perftest all/compulsory/cmd1;cmd2... timeout n n1;n2;n3... (pääohjelman toteuttama)	Ajaa ohjelmalle tehokkuustestit. Tyhjentää tietorakenteen ja lisää sinne <i>n1</i> kpl satunnaisia kaupunkeja (ks. <code>random_add</code>). Sen jälkeen arpoo <i>n</i> kertaa satunnaisen komennon. Mittaa ja tulostaa sekä lisäämiseen että komentoihin menneen ajan. Sen jälkeen sama toistetaan <i>n2</i> :lle jne. Jos jonkin testikierroksen suoritus aika ylittää <i>timeout</i> sekuntia, keskeytetään testien ajaminen (tämä ei välttämättä ole mikään ongelma, vaan mielivaltaisen aikaraja). Jos ensimmäinen parametri on <i>all</i> , arvotaan lisäyksen jälkeen kaikkista komennoista, joita on ilmoitettu kutsuttavan usein. Jos se on <i>compulsory</i> , testataan vain komentoja, jotka on pakko toteuttaa. Jos parametri on lista komentoja, arvotaan komento näiden joukosta (tällöin kannattaa mukaan ottaa myös <code>random_add</code> , jotta lisäyksiä tulee myös testikierroksen aikana). Jos ohjelmaa ajaa graafisella käyttöliittymällä, "stop test" nappia painamalla testi keskeytetään (nappiin reagointi voi kestää hetken).
testread 'in-tiedostonimi' 'out-tiedostonimi' (pääohjelman toteuttama)	Ajaa toiminnallisuustestin ja vertailee tulostuksia. Lukee komennot tiedostosta in-tiedostonimi ja näyttää ohjelman tulostuksen rinnakkain tiedoston out-tiedostonimi sisällön kanssa. Jokainen eroava rivi merkitään kysymysmerkillä, ja lopuksi tulostetaan vielä tieto, oliko eroavia rivejä.
help (pääohjelman toteuttama)	Tulostaa listan tunnetuista komennoista.
quit (pääohjelman toteuttama)	Lopettaa ohjelman. (Tiedostosta luettaessa lopettaa vain ko. tiedoston lukemisen.)

"Datatiedostot"

Kätevin tapa testata ohjelmaa on luoda "datatiedostoja", jotka `add`-komennolla lisäävät joukon kaupunkeja ohjelmaan. Kaupungit voi sitten kätevästi lukea sisään tiedostosta `read`-komennolla ja sitten kokeilla muita komentoja ilman, että kaupungit täytyisi joka kerta syöttää sisään käsin. Alla on esimerkki datatiedostosta, joka löytyy nimellä *example-data.txt*:

```
# Adding towns
add_town Hki Helsinki (3,0) 1000
add_town Tpe Tampere (2,2) 500
add_town Ol Oulu (3,5) 400
add_town Kuo Kuopio (6,3) 300
add_town Tku Turku (1,1) 30
# Adding vassalships
add_vassalship Ol Kuo
add_vassalship Kuo Hki
add_vassalship Tpe Hki
```


Esimerkki ohjelman toiminnasta

Alla on esimerkki ohjelman toiminnasta. Esimerkin syöte löytyy tiedostosta *example-in.txt* ja alla oleva tulostus tiedostosta *example-out.txt*. Eli voit testata esimerkin toimimista käynnistämällä ohjelman ja antamalla komennon *testread 'example-in.txt' 'example-out.txt'*.

```
> clear
Cleared all towns
> size
Number of towns: 0
> read "example-data.txt"
** Commands from 'example-data.txt'
> # Adding towns
> add_town Hki Helsinki (3,0) 1000
Helsinki: pos=(3,0), tax=1000, id=Hki
> add_town Tpe Tampere (2,2) 500
Tampere: pos=(2,2), tax=500, id=Tpe
> add_town Ol Oulu (3,5) 400
Oulu: pos=(3,5), tax=400, id=Ol
> add_town Kuo Kuopio (6,3) 300
Kuopio: pos=(6,3), tax=300, id=Kuo
> add_town Tku Turku (1,1) 30
Turku: pos=(1,1), tax=30, id=Tku
> # Adding vassalships
> add_vassalship Ol Kuo
Added vassalship: Oulu -> Kuopio
> add_vassalship Kuo Hki
Added vassalship: Kuopio -> Helsinki
> add_vassalship Tpe Hki
Added vassalship: Tampere -> Helsinki
>
** End of commands from 'example-data.txt'
> size
Number of towns: 5
> alphalist
1. Helsinki: pos=(3,0), tax=1000, id=Hki
2. Kuopio: pos=(6,3), tax=300, id=Kuo
3. Oulu: pos=(3,5), tax=400, id=Ol
4. Tampere: pos=(2,2), tax=500, id=Tpe
5. Turku: pos=(1,1), tax=30, id=Tku
> mindist
Turku: pos=(1,1), tax=30, id=Tku
> maxdist
Kuopio: pos=(6,3), tax=300, id=Kuo
> distlist
1. Turku: pos=(1,1), tax=30, id=Tku
2. Helsinki: pos=(3,0), tax=1000, id=Hki
3. Tampere: pos=(2,2), tax=500, id=Tpe
4. Oulu: pos=(3,5), tax=400, id=Ol
5. Kuopio: pos=(6,3), tax=300, id=Kuo
> nth_distance 2
Helsinki: pos=(3,0), tax=1000, id=Hki
> find Kuopio
```

```
Kuopio: pos=(6,3), tax=300, id=Kuo
> change_name Tpe Manse
Manse: pos=(2,2), tax=500, id=Tpe
> vassals Hki
1. Kuopio: pos=(6,3), tax=300, id=Kuo
2. Manse: pos=(2,2), tax=500, id=Tpe
> taxer_path Tpe
0.    Manse: pos=(2,2), tax=500, id=Tpe
1. -> Helsinki: pos=(3,0), tax=1000, id=Hki
> longest_vassal_path Hki
0.    Helsinki: pos=(3,0), tax=1000, id=Hki
1. -> Kuopio: pos=(6,3), tax=300, id=Kuo
2. -> Oulu: pos=(3,5), tax=400, id=Ol
> total_net_tax Tku
Total net tax of Turku: 30
> total_net_tax Ol
Total net tax of Oulu: 360
> total_net_tax Kuo
Total net tax of Kuopio: 306
> total_net_tax Hki
Total net tax of Helsinki: 1084
> remove Kuo
Kuo removed.
> vassals Hki
1. Oulu: pos=(3,5), tax=400, id=Ol
2. Manse: pos=(2,2), tax=500, id=Tpe
>
```